Multi-objective bandit algorithms with Chebyshev scalarization

Lawrence Mandow [a;*], **Sergio Martín-Albo**[b] **and Jose-Luis Perez-de-la-Cruz**[a]

[a]Universidad de Málaga, Andalucía Tech, Dpto. de Lenguajes y Ciencias de la Computación, Málaga, España
[b]ITIS Software, Universidad de Málaga

Abstract. In this paper we analyze several alternatives for Chebyshev scalarization in multi-objective bandit problems. The alternatives are evaluated on a reference bi-objective benchmark problem of Pareto frontier approximation. Performance is analyzed according to three measures: probability of selecting an optimal action, regret, and unfairness. The paper presents a new algorithm that improves the speed of convergence over previous proposals at least by one order of magnitude.

1 Introduction

Bandit problems [10] arose from the problem of resource allocation in clinical trials. They can be assimilated to the problem of optimally playing a *multi-armed bandit*. This device is a slot machine with several levers. Associated to each lever there is a (generally unknown) probability distribution of rewards. The goal is to maximize the expected reward after a given number of lever pushings.

Bandit problems can be seen as a simplified case of reinforcement learning [13] and their solution is related to well-known algorithms such as MCTS (Monte Carlo Tree Search) and its variants (e.g. UCT [4]).

Bandit problems can be generalized to multiobjective settings [5][14][8]. In this case, the reward of pushing a lever is not a scalar value, but a vector, and there are several (many) optimal values for the expected reward. All these values define the *Pareto frontier* of the problem. A possible strategy to approximate the Pareto frontier is to solve different scalarizations of the original, vectorial problem.

Drugan and Nowé [5] analyzed linear and Chebyshev scalarizations for multi-objective bandit problems. Linear scalarizations find only a subset of Pareto-optimal solutions (the so-called *supported solutions*), and this limitation is not suitable in certain situations. On the contrary, Chebyshev scalarization can (in principle) find any Pareto-optimal solution. However, experimental results shown in [5] concerning Chebyshev scalarization have, in our opinion, room for improvement.

* Financiado por Plan Propio de Investigación de la Universidad de Málaga - Campus de Excelencia Internacional Andalucía Tech. L. Mandow supported by project *IRIS* PID2021-122812OB-I00 (co-financed by FEDER funds). This research is partially supported by the Spanish Ministry of Science and Innovation, the European Regional Development Fund (FEDER), Junta de Andalucía (JA), and Universidad de Málaga (UMA) through the research projects with reference PID2021-122381OB-I00 and UMA20-FEDERJA-065. S. Martín-Albo supported by Beca de Iniciación a la Investigación para estudiantes de grado y máster, I Plan Propio de Investigación y Transferencia de la Universidad de Málaga, España.). Corresponding Author. Email: lmandow@uma.es

In this paper we review the application of Chebyshev scalarization to multiobjective bandit problems and propose new algorithms in order to improve previous performance. A previous version of this paper appears in [11].

The paper is organized as follows: in section 2 we present the main concepts related to bandit problems, their multi-objective generalization, and Chebyshev scalarization. In section 3 we present the algorithm proposed in [5] and some new variants. Section 4 describes an example and analyzes the performance of these variants. Finally, some conclusions are drawn and some possible continuations are suggested.

2 Antecedents

2.1 Bandit problems

Formally, let us consider a problem where there are k possible *actions* a_i, $i = 1 \ldots k$, each one associated to a probability distribution with its own mean μ_i. Distributions are stationary and independent. Every execution of a_i returns a value (reward) taken independently from its probability distribution. An agent (who doesn't know the distributions) must sequentially select N actions. A *policy* is a rule that determines the agent's next action.

Let μ^* be the greatest mean, corresponding to the optimal action. Executing a suboptimal action i has an associated *regret* $\Delta_i = \mu^* - \mu_i$. The agent's goal is to *minimize the expected regret* when executing N actions, or equivalently, to *maximize the expected reward*.

To this end, the agent must use actions both to estimate the expected value of each action, and to profit from the supposedly best action (the so-called exploration/exploitation dilemma). A greedy policy selects at each step the action whose estimation is greatest (exploitation), but in that way the agent doesn't improve the estimations of the other actions (exploration), whose expected rewards could be possibly better.

In fact, the greedy strategy is provably suboptimal. Lai and Robbins [9] proved that optimal regret grows asymptotically as the logarithm of the number N of actions. They also defined asymptotically optimal policies by associating an *upper confidence index* to each action and choosing the action with greatest index.

Agrawal [1] improved this result by defining asymptotically optimal policies that are simpler to compute. Then, Auer, Cesa-Bianchi, and Fischer [2] improved those previous results and defined policies that obtain a logarithmic regret uniformly distributed along time. The most used is the so-called UCB1 (Upper Confidence Bound 1) policy.

Algorithms 1 and 2 implement the UCB1 policy [2]. Algorithm 1 starts by executing each action once. Algorithm 2 computes in line 3 the average x_i of the rewards obtained by executing each action i. Next action is chosen in line 1 of algorithm 2. This action is the one with the greatest *upper bound*. The bound is the sum of average reward x_i and a confidence factor that decreases with the number n_k of times that i has been executed and increases with the total number t of executed actions. In this way, the x_i term favors exploitation while the confidence term favors exploration.

Algorithm 1 Algorithm UCB1. The method $execute(a)$ performs action a and returns the reward.

Input N, number of steps; K, number of actions; $\vec{a} = (a_1, \dots, a_K)$, vector of actions.

```
1: Define $\vec{x} = (x_1, \dots x_K), \vec{n} = (n_1, \dots n_K)$ vectors to keep average reward and step count for every action respectively.
2: for $i \in \{1..K\}$ do
3:     $x_i \leftarrow execute(a_i)$
4:     $n_i \leftarrow 1$
5: end for
6: $t \leftarrow K$
7: for $N$ steps do
8:     $stepUCB1(\vec{a}, \vec{x}, \vec{n}, t)$
9: end for
```

Algorithm 2 Method stepUCB1. Method $execute(a)$ performs action a and returns the reward.

Input $\vec{a}, \vec{x}, \vec{n}$, vectors of actions, average rewards and step counts for every action; t, global step count.

```
1: $i \leftarrow \arg\max_k (x_k + \sqrt{\frac{2 \ln t}{n_k}})$
2: $n_i \leftarrow n_i + 1$
3: $x_i \leftarrow x_i + \frac{1}{n_i}(execute(a_i) - x_i)$
4: $t \leftarrow t + 1$
```

2.2 *Multi-objective optimization*

Multi-objective optimization [7] is a generalization of scalar optimization in which a set of D different objectives are considered at once. We will assume that we want to maximize all objectives.

A multi-objective problem has a set $\mathcal{X}$ of *feasible solutions*; the reward vectors of each feasible solution form a set $Y \subset \mathbb{R}^D$. In the following, by abuse of language we will identify the set of solutions with the set Y of vectors.

The solution to a multi-objective problem is not given by a unique element. Given two vectors $\vec{u}, \vec{v} \in \mathbb{R}^D$, we say that $\vec{u}$ dominates $\vec{v}$ iff for each component i, $u_i \geq v_i$ and at least for one element j, $u_j > v_j$. This dominance relation is a partial order. The solution to a multi-objective problem is given by all its *Pareto optima*. A solution is Pareto optimal iff its value is not dominated by the value of any other solution. The set of Pareto optimal values defines the so-called *Pareto frontier*. Given a set Y of vectors, we will denote by $\mathcal{N}(Y)$ the subset of vectors in Y non dominated by any vector in Y.

Algorithms that try to find the set of Pareto optimal solutions to a problem are frequently direct generalizations of scalar optimization algorithms. The work of [5] described Pareto UCB1, a generalization of the UCB1 algorithm that uses the dominance preference relation to order reward vectors. Later, algorithm iPUCB [6] (improved Pareto Upper Confidence Bound) provided a multi-objective generalization of the improved UCB1 algorithm [3] which achieves better performance discarding dominated actions.

Another approach to multi-objective optimization is the computation of just one Pareto-optimal solution (or a small subset of the Pareto frontier). That unique solution is supposed to be the one that fits better the preferences of the decision maker. Sometimes it is possible to scalarize reward vectors, i. e., to associate a scalar value to each solution. Then we can apply single-objective optimization algorithms to solve the problem.

Chebyshev scalarization[1] belongs to a family of scalarization methods that try to minimize the distance between the vector solution and a reference point. A usual reference point is the *ideal point* $\vec{\alpha} = (\alpha_1, \dots \alpha_D)$, where α_i is the optimal value obtained in the scalar maximization of the i-th objective,

$$\alpha_j = \max_{\vec{y} \in Y} y_j \tag{1}$$

In Chebyshev scalarization, the distance to minimize is the *Chebyshev distance*, or norm l_∞, defined for a vector $\vec{y} = (y_1, \dots, y_D)$ as

$$\max_k \{w_k |y_k - \alpha_k|\} \tag{2}$$

where the weight vector $\vec{w} = (w_1, \dots w_D)$ denotes the preference of the decision maker.

A fundamental property of this approach is that for every non-dominated solution $\vec{y^*}$ there exists a set of weights such that $\vec{y^*}$ minimizes the distance to $\vec{\alpha}$ [12]. Analogously, given the solutions to

$$\begin{gathered} \text{Min } \max_k \{w_k |y_k - \alpha_k|\} \\ \text{Subject to } \vec{y} \in Y \end{gathered} \tag{3}$$

at least one of them is non-dominated.

So, by trying different vectors of weights, Chebyshev scalarization can be used to approximate a Pareto frontier.

In this paper we are mainly concerned with methods that choose the next action combining upper confidence bounds and Chebyshev scalarization. Nevertheless, in section 4, we shall test the performance of these methods on a previously proposed benchmark problem that seeks to approximate a Pareto frontier [5] using a series of such Chebyshev scalarizations. The algorithms and methods are described in section 3.

2.3 *Multi-objective bandits*

Bandit problems can be generalized by assuming that every action yields a reward vector, where each component represents a different objective to maximize. This problem was proposed in [5] and called MOMAB (Multi-Objective Multi-Armed Bandit problem). In this paper, we will try to follow their notation.

The application of several scalarization strategies to MOMAB is discussed in [5]. As we advanced in section 2.2, Chebyshev scalarization offers some advantages. Firstly, it is computationally efficient, since the dominance relation is easy to compute; and scalar algorithms can be easily adapted. Secondly, it can compute a Pareto-optimal solution suitable for the intuitive preferences expressed by the decision maker. Moreover, it allows the computation of every value in the Pareto frontier.

[1] Sometimes spelled Tchebycheff.

2.4 *Chebyshev scalarization for multi-objective bandits*

As we said in section 2.2, it is possible to find a vector that maximizes the vector reward by minimizing the distance between the reward vector to a reference point. In fact, in [5] algorithm UCB1 is proposed to solve the problem. However, UCB1 is a maximizing algorithm and in the problem at hand we want to minimize a function (the distance). For this reason, a reformulation is needed. A new reference point is defined: the so-called *nadir point* $\vec{z} = (z_1, \ldots, z_D)$ where

$$z_j = \min_{\vec{y} \in \mathcal{N}(Y)} y_j \tag{4}$$

(notice that the minimum is taken only over the points in $\mathcal{N}(Y)$).

Now we can maximize the distance to the nadir point, that is equivalent to minimizing the distance to the ideal point. In fact, in [5] it is proposed to substract a small positive value ϵ_j from each component of the nadir point.

In conclusion, considering the new reference point $\vec{z}$, we define the following weighted scalarization function:

$$f_T(\vec{y}) = \min_k \{w_k(y_k - z_k)\} \tag{5}$$

And now, given the solutions to the following problem:

$$\begin{aligned} &\text{Max } f_T(\vec{y}) \\ &\text{Subject to } \vec{y} \in Y \end{aligned} \tag{6}$$

at least one of them is a non-dominated vector.

With this reformulation it is possible to adapt algorithm UCB1 to solve problem 6. A detailed description is given in the following section.

3 Algorithms

An algorithm is proposed in [5] to approximate the Pareto frontier of a MOMAB by solving a set of scalar problems arising from the Chebyshev scalarization described above. The main ideas are the following:

- There is a set of S scalarization functions $f^1, \ldots f^S$, each one given by a different weight vector.
- The algorithm chooses randomly at each step one of these S functions, with uniform probability.
- Different estimations and counts are mantained for different actions. That means that S optimizations are carried out independently, one for each scalarization function.
- After scalarizating the estimations of an action, the confidence factor is added. This factor corresponds to the number of times in which that function was chosen and that action was executed.

These ideas are applied in algorithms 3 and 4. The procedure so defined will be called C0.

Algorithm 3 Algorithm C0. Method $execute(a)$ performs action a and returns the reward vector.

Input N, number of steps (decisions); K, number of actions; $\vec{a} = (a_1, \ldots, a_K)$, vector of actions; S, number of functions; $C = \{f^1, \ldots f^S\}$, set of functions (weights).

```
 1: Define $X$, an array $K \times S$ such that $X_{ij}$ denotes the vector average of obtained rewards for action $i$ when function $j$ was chosen.
 2: Define $N$, an array $K \times S$ such that $N_{ij}$ denotes the number of times in which action $i$ was executed when function $j$ was chosen.
 3: Define $\vec{t} = (t_1, \ldots t_S)$ such that $t_j$ denotes the number of times function $j$ was chosen.
 4: for $i \in \{1..K\}$ do
 5:     for $j \in \{1..S\}$ do
 6:         $X_{ij} \leftarrow execute(a_i)$
 7:         $N_{ij} \leftarrow 1$
 8:     end for
 9: end for
10: for $j \in \{1..S\}$ do
11:     $t_j \leftarrow K$
12: end for
13: for each step in $N$ do
14:     Choose uniformly at random a function $f^j$
15:     stepC0($j, \vec{a}, X, N, \vec{t}$)
16: end for
```

Algorithm 4 Method stepC0.

Input j, index of scalarizing function; $\vec{a}$; vector of actions; X, N, arrays that store the average of reward vectors, and the step counts; $\vec{t}$, global step count.

```
1: $i \leftarrow \arg\max_k (f_j(X_{kj}) + \sqrt{\frac{2 \ln t_j}{N_{kj}}})$
2: $N_{ij} \leftarrow N_{ij} + 1$
3: $X_{ij} \leftarrow X_{ij} + \frac{1}{N_{ij}}(execute(a_i) - X_{ij})$
4: $t_j \leftarrow t_j + 1$
```

This paper explores some improvements on algorithm C0:

1. Firstly, it is not necessary to perform S independent optimizations UCB1 (one for each scalarization function). We can average all rewards for action i, whichever function f^j was chosen. In this way, all scalarization functions benefit from all the values sampled for action i.
2. Secondly, remember that UCB1 adds the confidence factor directly to the estimation of each action. On the contrary, C0 adds the confidence factor to the *scalarization of the estimations* (algorithm 4, line 1). Perhaps it would be interesting to scalarize the bounds of the estimations instead of bounding the scalarization of the estimations.
3. Lastly, ties can eventually arise when choosing an action. At least one of them will correspond to a non-dominated point (see section 2.2). Therefore, we propose to break the tie favoring non-dominated estimations.

In conclusion, we consider in this paper three algorithms:

- C0: Shown above. It interleaves S independent UCB1 optimizations. Each function keeps an estimation of its own. The confidence factor is added to the scalarization of the estimation.

- C1: All functions share estimations (proposal 1). Confidence factor is added to the scalarization of the estimation. See algorithms 5 and 6.
- C2: All functions share estimations (proposal 1). Confidence factor is added to the estimation and the result is scalarized (proposal 2). This algorithm is the same as C1 (algorithm 5), but in line 10 it calls $stepC2$ (algorithm 7) instead of $stepC1$.

Proposal 3 is applied in all algorithms (C0, C1, and C2).

Algorithm 5 Algorithm C1. Method $execute(a)$ performs action a and returns the reward vector.

Input N, number of steps (decisions); K, number of actions; $\vec{a} = (a_1, \ldots, a_K)$, vector of actions; S, number of functions; $C = \{f^1, \ldots f^S\}$, set of functions (weights).

```
1: Define $X$, an array $K \times D$ such that $\vec{x_i}$ denotes the vector average of rewards for action $i$ ($i$-th row).
2: Define $\vec{n}$, a vector with $K$ components such that $n_i$ denotes the number of times action $i$ was executed.
3: for $i \in \{1..K\}$ do
4:     $\vec{x_i} \leftarrow execute(a_i)$
5:     $n_i \leftarrow 1$
6: end for
7: $t \leftarrow K$
8: for $N$ steps do
9:     Choose uniformly at random a function $f^j$
10:    stepC1$(j, \vec{a}, \vec{n}, X, t)$
11: end for
```

Algorithm 6 Method stepC1.

Input j, index of scalarizing function; $\vec{a}$, $\vec{n}$, vector of actions and vector of counts for each action; X, array that stores the average of reward vectors; t, global step count.

```
1: $i \leftarrow \arg\max_k (f^j(\vec{x_k}) + \sqrt{\frac{2\ln t}{n_k}})$
2: $n_i \leftarrow n_i + 1$
3: $\vec{x_i} \leftarrow \vec{x_i} + \frac{1}{n_i}(execute(a_i)) - \vec{x_i})$
4: $t \leftarrow t + 1$
```

Algorithm 7 Method stepC2.

Input j, index of scalarizing function; $\vec{a}$, $\vec{n}$, vector of actions and vector of counts for each action; X, array that stores the average of reward vectors; t, global step count.

```
1: $i \leftarrow \arg\max_k (f^j(\vec{x_k} + \sqrt{\frac{2\ln t}{n_k}}))$
2: $n_i \leftarrow n_i + 1$
3: $\vec{x_i} \leftarrow \vec{x_i} + \frac{1}{n_i}(execute(a_i)) - \vec{x_i})$
4: $t \leftarrow t + 1$
```

4 Experiments and analysis

In this section we execute the algorithms presented in section 3 on the problem proposed in [5], and compare their performances.

The problem has 20 actions and two objectives. Mean values for each action are $\mu_1 = (0.55, 0.5), \mu_2 = (0.53, 0.51), \mu_3 = (0.52, 0.54), \mu_4 = (0.5, 0.57), \mu_5 = (0.51, 0.51), \mu_6 = (0.5, 0.5)$ and $\mu_k = (0.48, 0.48), k = 7 \ldots 20$. The first four values are Pareto-optimal. In all cases rewards follow a Bernoulli distribution.

Eleven scalarization functions are considered, defined by weight vectors $(1, 0), (0.9, 0.1), \ldots (0.1, 0.9), (0, 1)$. The values ϵ_i substracted from the nadir point were all 0.01. Results shown correspond to the average of 250 different agents. Each agent executed 10^6 steps.

4.1 Performance measures

In these experiments three measures proposed in [5] were evaluated:

- Number of times (in percentage) that a Pareto-optimal action was chosen. This is a standard measure for Bandit problems.
- A measure of regret defined for scalarized algorithms. Assume that the optimal value for scalarization function j is

$$f_j^* = \max_{k \in \mathcal{A}} f_j(\mu_k) \tag{7}$$

Then the regret for choosing action i when using function j is:

$$\Delta_{ij} = f_j^* - f_j(\mu_i) \tag{8}$$

- Lastly, we consider a mesure of *unfairness* that was proposed in [5]. A shortcoming of the regret as performance measure is that when a policy exploits just one Pareto optimal solution and ignores the others, the regret can be small, but the approximation of the Pareto frontier is very poor. For this reason, an unfairness measure was proposed, that is the variance of the number of times each optimal action was chosen:

$$\sigma = \frac{1}{|\mathcal{A}^*|} \sum_{i \in \mathcal{A}^*} (t_i - T)^2 \tag{9}$$

where $\mathcal{A}^*$ is the set of Pareto-optimal actions, t_i is the number of times action i was executed, and T is the average of these numbers.

4.2 Results

Figs. 1, 2 and 3 show the results of the experiments with algorithms C0, C1, and C2 as a function of the number of steps. Measures were taken every 500 steps. Fig. 1 displays the evolution of the probability of choosing a Pareto-optimal action. Fig. 2 displays the average accumulated regret. Fig. 3 shows the evolution of the unfairness measure.

4.3 Discussion

The worst results are obtained for algorithm C0. Since eleven functions are independently optimized, convergence is slow. It can be conjenctured that the more functions to consider, the slower the convergence.

Results show that sharing value estimations by different functions accelerates convergence. More concretely, Fig. 1 shows that C1 obtains with 10^5 steps the percentage of optimal selections that C0 obtains with 10^6 steps. In all cases, the oscillation of average values is typical for the Bernoulli distributions followed by the rewards.

Moreover, Fig. 1 shows that C2 converges to optimal solutions faster than C0 and C1. Fig. 2 confirms this point: the smallest values of regret are achieved by C2, followed by C1 and lastly C0.

Concerning unfairness, Fig. 3 also shows that the performance is different for different algorithms. In the initial steps of learning, when actions are chosen practically at random, we can suppose that unfairness will be small. However, as algorithms find out some Pareto optima earlier than others, we can expect that unfairness will increase. Only when all Pareto optima have been found, unfairness will

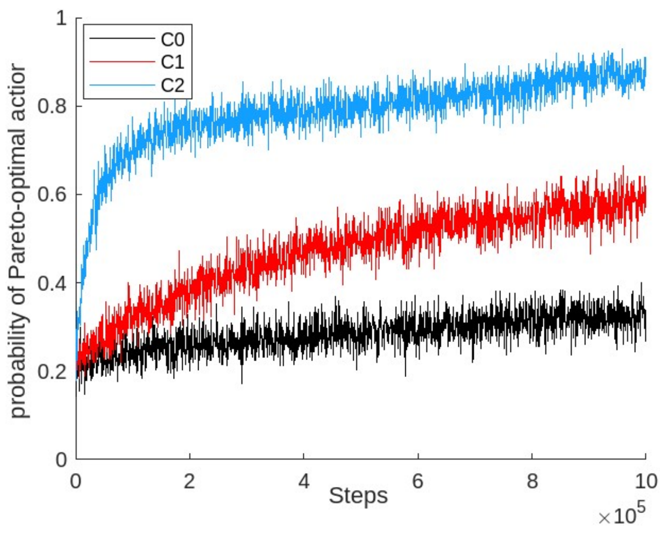

Figure 1. Probability of choosing a Pareto-optimal action

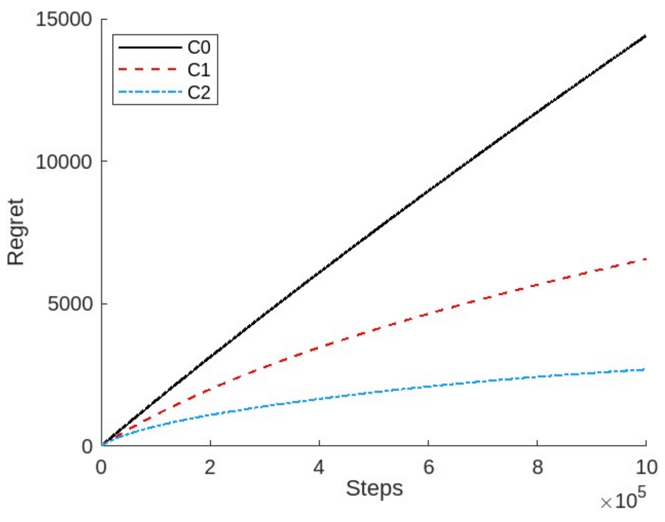

Figure 2. Average regret

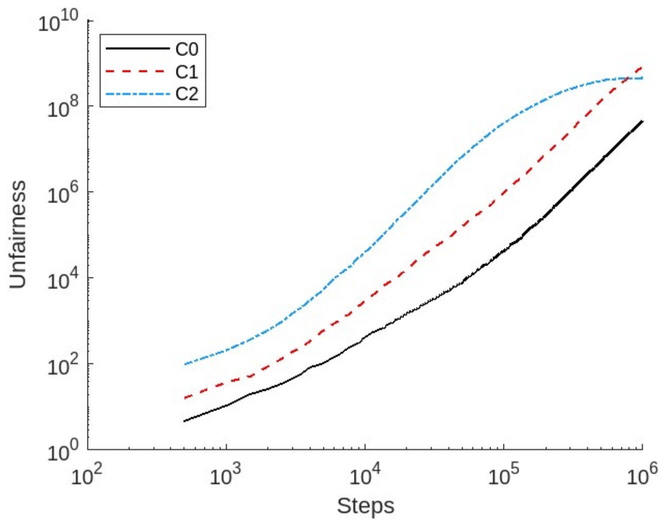

Figure 3. Average unfairnesss (log-log scale)

decrease. These intuitions are confirmed in Fig. 3. Algorithm C2 begins to decrease in unfairness after 10^5 steps, when it has achieved a reasonable degree of convergence. Algorithm C1 presents at the begining a smaller unfairness, since its convergence is slower; but finally its unfairness is greater and after 10^6 steps it has converged. Algorithm C0 shows a graph almost one order of magnitude below C1 but also increasing. This behavior is consistent with the slower convergence displayed in Fig. 1.

To sum up, results show that the new algorithm C2 presented in this paper can represent a significant improvement over C0 and C1. Concretely, for the reference problem defined in [5] the improvement is of one order of magnitude.

5 Conclusions and future works

This paper addresses the multi-objective multi-armed bandit problem (MOMAB). More concretely, we review methods based on Chebyshev scalarization. These methods can be used both to find one solution and to approximate the whole Pareto frontier. An intuitive variant of the algorithm in [5] (algorithm C0) is presented (algorithm C1). A reformulation of the criterion for action selection is also proposed (algorithm C2), based on the idea of bounding reward estimations (as in the UCB1 algorithm). In this way the statistical meaning of this bound and of its confidence factor are preserved. Considering the speed of convergence to the Pareto frontier, the evaluation on a reference problem shows that the performance of C2 improves those of C0 and C1 at least in one order of magnitude.

Future work includes a more extensive evaluation of algorithm C2. While the main focus of this paper was on methods to choose the next action combining upper confidence bounds and Chebyshev scalarization, these were tested on a Pareto-frontier approximation problem. The encouraging results obtained make a comparison of C2 with Pareto algorithms like those described in [5] [6] an interesting line of future research. Also more generally, the scalarization method here described could be applied to other problems of multi-objective reinforcement learning.

References

[1] R. Agrawal, ‘Sample mean based index policies with O(logn) regret for the multi-armed bandit problem’, *Advances in Applied Probability*, **27**, 1054–1078, (1995).

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer, ‘Finite-time analysis of the multiarmed bandit problem’, *Mach. Learn.*, **47**(2-3), 235–256, (2002).

[3] Peter Auer and Ronald Ortner, ‘UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem’, *Period. Math. Hung.*, **61**(1-2), 55–65, (2010).

[4] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton, ‘A survey of monte carlo tree search methods’, *IEEE Trans. Comput. Intell. AI Games*, **4**(1), 1–43, (2012).

[5] Madalina M. Drugan and Ann Nowé, ‘Designing multi-objective multi-armed bandits algorithms: A study’, in *The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013*, pp. 1–8. IEEE, (2013).

[6] Madalina M. Drugan, Ann Nowé, and Bernard Manderick, ‘Pareto upper confidence bounds algorithms: An empirical study’, in *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, ADPRL 2014, Orlando, FL, USA, December 9-12, 2014*, pp. 1–8, (2014).

[7] Matthias Ehrgott, *Multicriteria Optimization*, Springer, 2005.

[8] Alihan Hüyük and Cem Tekin, ‘Multi-objective multi-armed bandit with lexicographically ordered and satisficing objectives’, *Mach. Learn.*, **110**(6), 1233–1266, (2021).

[9] T.L Lai and Herbert Robbins, ‘Asymptotically efficient adaptive allocation rules’, *Advances in Applied Mathematics*, **6**(1), 4–22, (1985).

[10] Tor Lattimore and Csaba Szepesvari, *Bandit algorithms*, Cambridge University Press, 2020.

[11] Lawrence Mandow, Sergio Martín-Albo, and Jose-Luis Perez-de-la-Cruz, ‘Algoritmos para el problema del bandido multi-objetivo basados en la escalarización de Chebyshev’, in *XIX Conferencia de la Asociación Española para la Inteligencia Artificial CAEPIA 20/21, 22-24 septiembre 2021, Málaga (España)*, pp. 135–140, (2021).

[12] Kaisa Miettinen, *Nonlinear multiobjective optimization*, Springer, 1998.

[13] Richard Sutton and Andrew Barto, *Reinforcement learning: an introduction*, The MIT Press, 2nd edn., 2018.

[14] Yinglun Zhu and Robert Nowak, ‘On regret with multiple best arms’, in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, eds., Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, (2020).